Scientific Programming 22 (2014) 273–283
DOI 10.3233/SPR-140393
IOS Press

Tools and methods for measuring and tuning the energy efficiency of HPC systems

Robert Schöne [a,*], Jan Treibig [b], Manuel F. Dolz [c], Carla Guillen [d], Carmen Navarrete [d], Michael Knobloch [e] and Barry Rountree [f]

[a] *Center for Information Services and High Performance Computing (ZIH), Technische Universität Dresden, Dresden, Germany*
E-mail: robert.schoene@tu-dresden.de
[b] *Erlangen Regional Computing Center, University Erlangen-Nuremberg, Erlangen, Germany*
E-mail: jan.treibig@fau.de
[c] *Department of Informatics, Universität Hamburg, Hamburg, Germany*
E-mail: manuel.dolz@informatik.uni-hamburg.de
[d] *Leibniz Rechenzentrum (LRZ) Bayerischen Akademie der Wissenschaften, München, Germany*
E-mails: {carla.guillen, carmen.navarrete}@lrz.de
[e] *Jülich Supercomputing Centre (JSC), Forschungszentrum Jülich GmbH, Jülich, Germany*
E-mail: m.knobloch@fz-juelich.de
[f] *Center for Applied Scientific Computation, Lawrence Livermore National Laboratory, Livermore, CA, USA*
E-mail: rountree@llnl.gov

Abstract. Energy costs nowadays represent a significant share of the total costs of ownership of High Performance Computing (HPC) systems. In this paper we provide an overview on different aspects of energy efficiency measurement and optimization. This includes metrics that define energy efficiency and a description of common power and energy measurement tools. We discuss performance measurement and analysis suites that use these tools and provide users the possibility to analyze energy efficiency weaknesses in their code. We also demonstrate how the obtained power and performance data can be used to locate inefficient resource usage or to create a model to predict optimal operation points. We further present interfaces in these suites that allow an automated tuning for energy efficiency and how these interfaces are used. We finally discuss how a hard power limit will change our view on energy efficient HPC in the future.

Keywords: Energy efficiency, energy and performance measurement, HPC, high performance computing, energy optimization tools, energy models, energy-efficiency tuning

1. Introduction to energy efficiency considerations in HPC

Within the last decade, energy costs of High Performance Computing (HPC) systems have significantly increased and now represent a significant share of the total cost of ownership (TCO) of such a system. Thus, servers and HPC systems are now not only evaluated in terms of throughput, but also in terms of energy efficiency as a second major requirement. In order to improve energy efficiency, research and developments targeting a lower power consumption have intensified with the ultimate goal to reach a maximum throughput within a given energy budget. The developments include the addition of hardware and operating system support for energy efficient operation in processors, node devices and memory. Current HPC systems are built from components with this kind of energy efficiency support and provide interfaces to control them. However, tuning energy efficiency is still a matter of research. HPC poses additional challenges regarding energy efficiency, as the impact on performance has to be very low in order to be acceptable. Higher performance and low power consumption are contradictory when a decrease in power consumption results in a

[*] Corresponding author: Robert Schöne, Center for Information Services and High Performance Computing (ZIH), Technische Universität Dresden, 01062 Dresden, Germany. E-mail: robert.schoene@tu-dresden.de.

longer execution time. Still, more and more HPC centers use the facilities to manipulate the energy consumption of applications e.g., by reducing the processor voltage and frequency. To find an optimal balance between energy efficiency and high performance of computations, hardware and software characteristics have to be taken into account.

The field of power and energy consumption analysis of HPC systems and applications is currently dominated by administrators and researchers, with very few application developers caring about these topics. However, with more and more HPC systems providing power measurement capabilities, HPC centers are able to switch from a CPU hour based allocation of resources to an allocation and accounting scheme that also reflects power bounds and electricity costs. Energy-aware scheduling has been detailed in [24,49,50] and is already supported by LoadLeveler [5]. Thus, in the near future all users of HPC systems might have to deal with energy efficiency, for which we provide an overview on several key areas.

The rest of the paper is organized as follows. We discuss several energy-efficiency metrics for both HPC systems and applications in Section 2. To complement the discussion on the basic steps prior to tuning, we provide guidelines for the choice of power measurement tools and interfaces in Section 3 and Section 4. The inclusion of performance and energy data into existing performance measurement infrastructures is described in Section 5. The paper is enriched with the illustration of an energy model that describes how the usage of power saving mechanisms influences performance and energy-efficiency in Section 6. We present tuning approaches and the integration of tuning cycles in performance analysis tools in Section 7. We close this paper with an outlook to the upcoming challenge of optimizing performance under a constant power consumption in Section 8 and provide a summary and outlook in Section 9.

2. Energy efficiency metrics

As in every optimization process, energy efficiency optimizations are highly influenced by the applied performance metric. While traditional performance metrics focus solely on throughput (e.g., FLOPS, IPC, MB/s), energy efficiency metrics focus on a trade-off between throughput (i.e., runtime) and energy costs (i.e., consumed energy). In this section, we discuss the most prominent metrics, the reader needs to know when interpreting energy-efficiency-performance data.

In HPC, the most common and known metric is FLOPS/W and depicts how efficient a system can execute the LINPACK benchmark, which is known to be very energy intensive. This metric is associated with the Green500 list [14] that was announced to accompany the established TOP500 list [13]. A major drawback of using LINPACK as the only benchmark case is that it does not reflect the typical workload of production HPC systems. The SPEC OMP2012 [33] benchmarks allow to accompany performance results with power and energy information. SPEC provides an infrastructure to measure the power consumption and defines measurement rules and certified watt-meters to avoid the submission of inaccurate results.

Energy-to-solution is an established metric to quantify the energy efficiency on a system for a specific code solving a specific problem. However, this metric only reflects the energy budget of running the application. The aim to either increase throughput within a given energy budget or reduce energy for a given throughput is not reflected. Thus, other metrics emphasize both runtime and energy consumption of an application. One metric in this context is the energy-delay product (EDP) introduced in 1994 [21]. It presents a compromise between low power and high performance and is well suited to guide research efforts. For scenarios where performance is even more important, the delay part may be emphasized by using an $\mathrm{ED}^n\mathrm{P}$ metric [10], often $\mathrm{ED}^2\mathrm{P}$ is used, or the generalized FTTSE metric [9], where an arbitrary function defines the ratio of performance and energy consumption.

In a computing center context though, energy efficiency is only one of the requirements. Additional constraints may be maximum power capping or constant power consumption with only small perturbation. If those constraints are met, a power supplier can often provide a significantly lower price, thus increasing the efficiency in terms of TCO. In Section 8 we discuss the challenges that arise with such a hard power bound.

3. Measuring power and energy

Prior to tuning energy consumption, researchers have to select an appropriate measuring tool with the required spatial and temporal granularity. In this section we discuss different solutions for measuring power and energy, accompanied by an overview of advantages and shortcomings for possible use cases. The following devices present an overview from finer to coarser spatial granularities of common power measurement alternatives.

RAPL [37], APM [26], and various performance counter based approaches (e.g. [11,18,25,43]) implement power consumption or energy consumption models based on processor events in hardware or software. One advantages of using such a model is the high update frequency (e.g., 1 kHz for RAPL), which is suitable for understanding the behavior of short code paths. However, there are several disadvantages when using processor event counters. These include for example measurement overhead due to the in-band measurement and model inaccuracies.

Power meter toolkits also allow measurements of fine granular devices such as CPU, disk, or different voltage lanes with dedicated current or power sensors. Examples of tools interfacing such sensors are PowerPack [15], PowerMon [8] and Amester [27]. High measurement frequencies of up to 1 kHz make them appropriate for code paths instrumentation and longer running applications. One disadvantage reported by Diouri et al. [12] are the differences in the accuracy of the measurements (more than 50% variation). The paper concludes that these tools require careful calibration.

At node level, common interfaces include paddle cards and power supply units (PSU) that provide Intelligent Platform Management Interface (IPMI) sensors. The temporal granularity is mostly within the range of 1–10 Hz [19,22]. Thus, they can only be used to create statistical profiles for long running applications, but will fail to examine short code paths. Hackenberg et al. [19] report accurate results from IPMI PSU measurements for constant power consumption scenarios. However, they also show that such sensors can be inaccurate if the running workflow provides a high power fluctuation.

There are "smart" Power Distribution Units (PDUs) which also feature a built-in powermeter circuitry capable of reporting out-bound measurements of instantaneous power via IPMI. These PDUs usually allow power readings on node level, rack level, or for the networking equipment. Typically, the queries can be done at the scale of seconds [19] or minutes [35]. Due to the low temporal resolution, they are only useful for obtaining statistical information.

Due to the diversity of tools, we only provide an overview. If one targets to lower the energy consumption of a specific component (e.g., main memory, CPU, or accelerator card) or a certain code-path, a fine grained instrumentation should be chosen. To verify optimizations at application level, the power should be measured at least at node level. The most important considerations when choosing a power measurement infrastructure include accuracy, and requirements on time and spatial resolutions.

4. Energy-efficiency related performance data

Measuring the power consumption is not necessarily sufficient to analyze applications and systems. To explain why and where a certain amount of energy has been spent when running an application, additional data is necessary. A low power consumption, for example, can be achieved by using a low frequency, but it could also be caused by the use of idle states, thermal throttling, or instructions that use less power. A high power consumption of a computing node can result from an intense use of an accelerator or the processor. Thus, the first step is to obtain additional data to understand the reasons for a certain power level and in the next step to lower the energy consumption.

Different aspects influence the power consumption of processors. These include the ACPI standardized P-, C-, and T-states [2], utilization of execution units and data transfers [32], and the clock gating of different parts of processors and processor cores [26]. While some of them are transparent (small scale clock gating), others can be observed via software and hardware counters. P-States that refer to voltage and frequency changes can be observed via instrumentation on methods that issue a change [39], via hardware performance monitoring units (PMUs) or special model specific registers (MSRs) like `APERF` and `MPERF` on newer x86 processors. C-States that implement idle-states can also be measured via instrumentation [39] and specialized MSRs [7] (residency counters). On recent Intel and AMD processors, T-States that refer to a processor throttling due to thermal constraints can be measured via MSRs as well as the temperature of computing cores. The Linux operating system allows to access most of this information in terms of special virtual file systems as `sysfs` and `procfs`.

C-States for example are measured on a per CPU base where each C-state issue is counted. Statistics about the C-state usage are reported in the `sysfs` on a per-CPU base. Due to C-state undemotion these numbers are, however, not entirely correct but represent only the issued C-states, not the ones that were actually initiated by hardware. Access to the hardware information is provided by residency counters [23] that can be accessed via the `msr` kernel module. A higher level of abstraction provides the powertop tool that provides statistics based on MSR readings.

Data transfers and computations in processor cores influence the power consumption to a high degree [32]. Many relevant performance events about processor core and cache activities and can be extracted from the hardware PMUs available on all modern processors. PMU counters that define activities within a processor core are often available per core or per thread (if simultaneous multi threading is active). Counters related to shared parts of the chip (often referred to as uncore by Intel or northbridge by AMD) require to program special PMUs which are valid on a per chip scope. To access the PMU data hardware performance monitoring tools like perf (included in the Linux kernel) or LIKWID [46] may be used.

5. Performance analysis tools and energy efficiency metrics integration

In the previous section we described how one can measure energy and performance related metrics. In this section, we present current state-of-the-art HPC performance analysis tools that integrate such information. These tools allow the measurement and comparison of the energy efficiency of parallel applications and relate energy information with performance data. Thus, they allow performance analysts to understand the reasons for an inefficient or efficiency energy consumption and what bottlenecks different application regions (e.g., subroutine calls) face.

A common way to observe the resource usage of an application is using hardware performance counters like floating-point instructions or cache misses via an interface like PAPI [44]. Most common HPC performance measurement tools (e.g., [1,16,17,29,30,36,40, 42,46]) allow performance counter measurements at a region level or a sampling of performance counters. As discussed in Section 2, several researchers build power and energy models based on hardware counters which would allow to estimate the energy consumption of such software regions. However, performance counter based models have to be determined for every new hardware generation and even for each processor or processor core due to process variation which can be unfeasible. Thus, to support power and energy measurements the performance tools need to include support for power meters and relevant data.

Instrumentation-based tracing tools record all events in an application (e.g. function enter/exit, MPI communication, etc.) and stores that in a trace file which can be analyzed post mortem, either manually with tools like Vampir or Paraver or automatically with Scalasca. Tracing provides a very high overhead and generates enormous amounts of data, especially when hardware counters are recorded. Instrumentation-based profilers obtain usually the same events as the tracing tools, but aggregate them instead of storing each event.

In summary, with profiling it is possible to see how much time was spent in a function and how often it was called, but only with tracing is it possible to examine each execution of the function.

5.1. Profiling-based tools

Power consumption data is often provided asynchronously to the application instrumentation, thus it can not necessarily be related to a certain code region. Additionally the power measurement is often implemented out-of-band to avoid interfering with the performance measurement. In such a case, power data is collected after the instrumented application finishes its execution. While this post-mortem integration is not necessarily a problem for tracing, it is for profiling.

Ideally, profiling tools access energy counters. These counters integrate the power data transparently to the measurement system and return the energy consumption between two measurement points. However, only few systems provide such an interface. Intel's RAPL interface that we discussed in Section 3 can be a solution for systems with Intel processors even though it is limited [19]. The alternative would be a power measurement sampling between the two measurement points by the tools and an integration in software, but this again implies an inherent overhead.

The profiling tool Periscope implements an interface to measure energy in Sandy Bridge-EP microarchitectures via the enopt library [34]. The current version of this library allows to measure the energy consumed at the instrumented regions of the user application.

5.2. Tracing-based tools

Tracing-based tools do not need the energy information at runtime but can merge this information into the trace in a post-mortem step. They also do not rely on energy data that is provided at the same time scale as the instrumentation points. Thus, there is a variety of tools supporting power and energy information. Schöne et al. [39] describe an plugin counter interface for VampirTrace [29] that can be used to include external information in application traces. They also present the inclusion of power relevant data, like frequency changes, the use of idle states and power consumption.

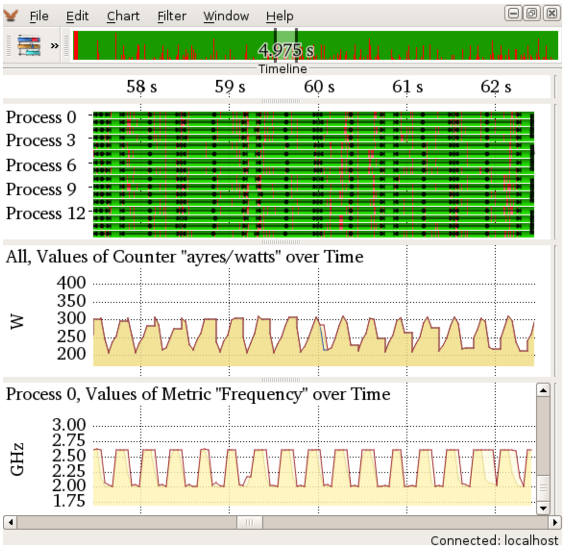
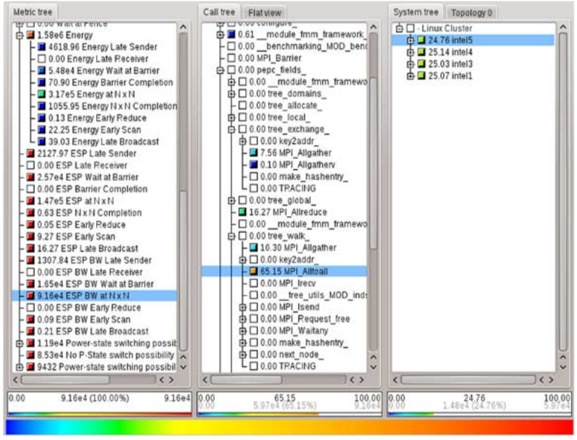


Fig. 1. Vampir: Displaying power and frequency information for an DVFS optimized execution of the MPI parallel NPB benchmark SP on a dual-socket Sandy Bridge system. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-140393.)

Figure 1 shows an example where power consumption and processor frequency is plotted with application runtime characteristics. Hackenberg et al. [19] and Knobloch et al. [27] implement counter plug-ins include other power sources allowing for fine-grained power measurements on IBM POWER7 hardware.

The interface for including external data into application traces has been ported to the Score-P measurement system and is included in the current release. The Technische Universität Dresden is implementing plug-ins for several power measurement devices so that experiments can be performed on power-aware HPC-systems.

Knobloch et al. [28] have analyzed the external power information in Vampir to see that a parallel application has shown high power consumption in MPI routines due to active waiting, i.e. polling with highest frequency whether the communication partner is ready. Using this information, Scalasca has been extended to determine the energy-saving potential in wait-states of parallel applications. An example of such an analysis is shown in Fig. 2. However, due to the aggregation of the data, Scalasca requires energy consumption data (similar to the profiling tools requirements) to perform a meaningful analysis. With such an energy data source available, it would be possible to determine whether it is better to do a race-for-idle or slow down computation to reduce wait times.



```
1.58e6 Energy
  4618.96 Energy Late Sender
  0.00 Energy Late Receiver
  5.48e4 Energy Wait at Barrier
  70.90 Energy Barrier Completion
  3.17e5 Energy at N x N
  1055.95 Energy N x N Completion
  0.13 Energy Early Reduce
  22.25 Energy Early Scan
  39.03 Energy Late Broadcast
2127.97 ESP Late Sender
0.00 ESP Late Receiver
2.57e4 ESP Wait at Barrier
0.00 ESP Barrier Completion
1.47e5 ESP at N x N
0.63 ESP N x N Completion
0.05 ESP Early Reduce
9.27 ESP Early Scan
16.27 ESP Late Broadcast
1307.84 ESP BW Late Sender
0.00 ESP BW Late Receiver
1.65e4 ESP BW Wait at Barrier
9.16e4 ESP BW at N x N
0.00 ESP BW Early Reduce
0.09 ESP BW Early Scan
0.21 ESP BW Late Broadcast
1.19e4 Power-state switching possit
8.53e4 No P-State switch possibility
9432 Power-state switching possibil

0.00    9.16e4 (100.00%)    9.16e4
```

```
0.61 __module_fmm_framework_
0.00 __benchmarking_MOD_benc
0.00 MPI_Barrier
0.00 pepc_fields_
  0.00 __module_fmm_framewo
  0.00 tree_domains_
  0.00 tree_allocate_
  0.00 tree_local_
  0.00 tree_exchange_
    0.00 key2addr_
    7.56 MPI_Allgather
    0.10 MPI_Allgatherv
    0.00 make_hashentry_
    0.00 TRACING
  0.00 tree_global_
  16.27 MPI_Allreduce
  0.00 __module_fmm_framewo
  0.00 tree_walk_
    10.30 MPI_Allgather
    0.00 key2addr_
    65.15 MPI_Alltoall
    0.00 MPI_Irecv
    0.00 __tree_utils_MOD_ind:
    0.00 MPI_Isend
    0.00 MPI_Request_free
    0.00 MPI_Waitany
    0.00 make_hashentry_
    0.00 next_node_
    0.00 TRACING

0.00    65.15    100.00
0.00    5.97e4 (65.15%)    9.16e4
```

```
- Linux Cluster
  24.76 intel5
  25.14 intel4
  25.03 intel3
  25.07 intel1

0.00    24.76    100.00
0.00    1.48e4 (24.76%)    5.97e4
```

Fig. 2. Scalasca: Displaying energy-saving potential (ESP) of PEPC, a plasma physics application from JSC, on an Intel Nehalem based cluster. The left pane shows the metric tree, here the ESP in collective MPI communication. The middle pane shows the distribution of these properties on the call tree, we see that 65.1% of the ESP is in a call to MPI_Alltoall in the tree_walk subroutine, and on the right pane the distribution among the system tree is shown, which indicates an equal distribution of the ESP across the nodes of the cluster. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-140393.)

Alonso et al. [4] present a framework which leverages Extrae and Paraver to analyze the power consumption and the energy consumption of parallel MPI and/or multithreaded scientific applications. The framework includes the `pmlib` library [6], which offers a simple interface to interact with a variety of wattmeters. Servat et al. [41] extend Extrae and Paraver to include power consumption information into traces from the RAPL energy model provided by current Intel processors.

5.3. *Use case*: *Detection of inefficient wait methods*

In [7] Barreda et al. define power sinks as a disagreements between the application activity and the system power consumption during the execution. They also present an inspection tool, based on Extrae + Paraver, that automatically detects power sinks by conducting a direct comparison between the application performance trace and the C-state traces per core.

In order to illustrate the possibilities of the inspection tool, Barreda et al. use an example corresponding to the concurrent solution a sparse linear system using ILUPACK,[1] a package that implements multilevel preconditioners for general and Hermitian posi-

[1] http://ilupack.tu-bs.de.

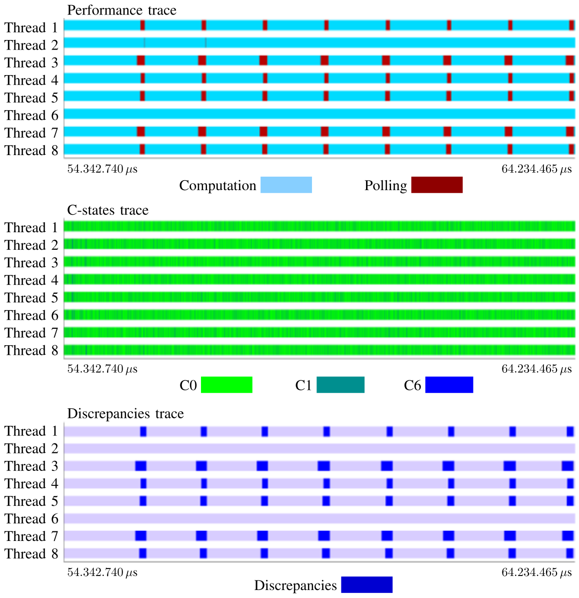


Fig. 3. Performance (*top*), C-states (*middle*) and discrepancies (*bottom*) trace, visualized with Paraver, for the concurrent execution of ILUPACK. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-140393.)

tive definite problems. The parallelization of this solver for multicore platforms in [3] relies on a task partitioning of the sparsity graph associated to the coefficient matrix of the system that yields a task acyclic graph. At runtime, tasks that are ready for execution (i.e, all dependencies are fulfilled) are added to a centralized queue. When a thread finishes its work, it polls to check if there is a new task in this queue.

Figure 3 shows fragments of the obtained performance and C-states traces for the parallel iterative solution stage of ILUPACK on a dual socket Nehalem system. The performance trace of the figure (top plot) shows that initially all threads perform computations. At regular intervals, threads wait for new tasks. The C-state trace (middle plot) does not show any significant variation. Thus, we can conclude that the threads perform a busy-wait. This is captured by the power-sink inspector, in the *power-sink* trace (bottom plot), which indicates a period where the application has performed no useful computation, but the cores remained active.

6. Using power models to predict energy efficient operation points

The ability to measure power enables the connection of the energy consumption to other parameters as code characteristics, runtime settings and frequency. In this section we describe how we can use the input from Section 5 for a simple qualitative power model that couples performance and energy to solution. We apply this model to the scalable `dgemm` benchmark. We use the ECM-model introduced in [45] and further refined and accompanied with a qualitative power model in [20]. In addition to previous publications we present the results by plotting energy to solution versus performance. This novel visualization technique has been introduced in [48].

The following basic assumptions go into the power and performance model:

(1) The whole N_c-core chip dissipates a certain baseline power W_0 when powered on, which is independent of the number of active cores and of the clock speed.[2]
(2) An active core consumes a dynamic power of $W_1 f + W_2 f^2$. We consider deviations from the baseline clock frequency f_0 such that $f = (1 + \Delta\nu) f_0$, with $\Delta\nu = \Delta f / f_0$.
(3) At the baseline clock frequency, the serial code under consideration runs at performance P_0. As long as there is no bottleneck, the performance is linear in the number of cores used, t, and the normalized clock speed, $1 + \Delta\nu$. The latter dependence will not be exactly linear if some part of the hardware (e.g., the off-core cache) runs in its own frequency domain. In presence of a bottleneck (like, e.g., memory bandwidth), the overall performance with respect to t is capped by a maximum value P_{roof}:

$$\begin{aligned} P(t) &= \min\bigl((1 + \Delta\nu) t P_0, P_{\mathrm{roof}}\bigr) \\ &= \min(t P_0 f / f_0, P_{\mathrm{roof}}). \end{aligned} \tag{1}$$

Since time to solution is inverse performance, the energy to solution becomes

$$E = \frac{W_0 + (W_1 f + W_2 f^2) t}{\min(t P_0 f / f_0, P_{\mathrm{roof}})}. \tag{2}$$

This model shows that any increase in performance (P_0 or P_{roof}) leads to proportional savings in energy to solution. Performance is thus the first-order tuning parameter for minimum energy. For the measurements in

[2] As can be seen in Fig. 4 W_0 is actually a function of the number of active cores. However, the qualitative insights from the model remain unchanged.

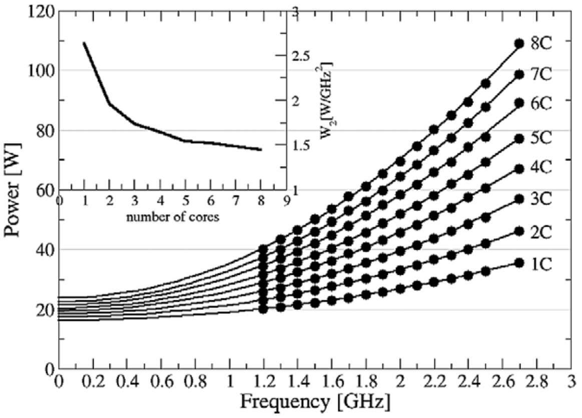


Fig. 4. Power for dgemm using different frequencies and core counts for single chip measurements. The filled circles are measured, the corresponding lines are fit functions determining the model parameters W_0, W_1 and W_2. The W_0 value varies from 16–23 W depending on the number of cores. This is due to the fragile fit on the left tail as there are no measurements below 1.2 GHz and also due to different states of all cores for different core counts. W_1 had no significant contribution and was therefore neglected on this test system. The resulting values for W_2 are shown in the inlet as function of number of cores. The TDP for this chip is 130 W.

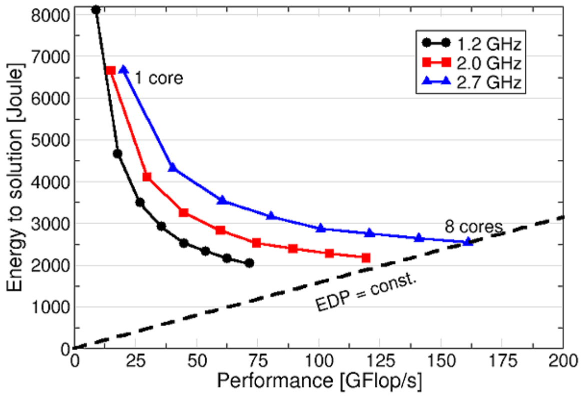


Fig. 5. Modelled power and performance. The modelled results predict correctly the optimal frequency for minimum EDP for dgemm. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-140393.)

this paper only the chip baseline power is taken into account neglecting any other power contribution on the node level. For more realistic estimates it is crucial to take into account the complete node baseline power. As demonstrated in [20] it is possible to analytically derive the frequency f_{opt} from Eq. (2) for minimal energy to solution:

$$f_{\text{opt}} = \sqrt{\frac{W_0}{W_2 t}}. \tag{3}$$

A large baseline power W_0 forces a large clock frequency to "get it over with" ("clock race to idle"). If considering the overall baseline power on current compute nodes (100 W on our test system) this case applies in many cases.

For qualitative results it is sufficient to assume approximate values that reflect general region properties known from code analysis and performance modeling (memory-boundedness, SIMD vectorization, pipeline utilization). For the test case covered in this paper the parameters were measured using RAPL on a 2.7 GHz Sandy Bridge processor. The chip power is measured for different core counts and frequencies and W_0, W_1, and W_2 are determined by fitting function to the measured points. Figure 4 shows the results for dgemm, for which on this particular chip there was a diminishing linear factor, thus W_1 was neglected. The inlet of Fig. 4 illustrates that the model parameter W_2 is a function of cores used per chip. Due to limited accuracy of the function fit and influence of C states on different cores also W_0 varies with core count.

With the model parameters determined energy to solution can be computed with core count and frequency as inputs. A useful way to visualize the relation between energy to solution and performance is to plot energy to solution versus performance with the number of cores used as a parameter within a data set for a specific frequency. One advantage of this plot variant is that points of constant EDP are straight lines. The targeted operation area is in the lower right quadrant defining the optimization space for a given code. For dgemm Fig. 6 shows that the lowest EDP is achieved if running at the highest nominal frequency. Still the turbo mode frequency is nearly as good in terms of EDP and better than lower frequencies. The model provided results shown in Fig. 5 correctly predict the optimal frequency setting. It enables qualitative insights in the energy to solution behavior of application codes. This model may be used in tuning tools Section 7 to determine the optimal operating point in terms of frequency and number of cores used surpassing solutions setting the frequency based on simple heuristics or generic settings.

7. Using performance analysis tools for auto-tuning energy efficiency

In this section we present how performance measurement infrastructures can be used to tune for energy efficiency. The advantage of combining analysis

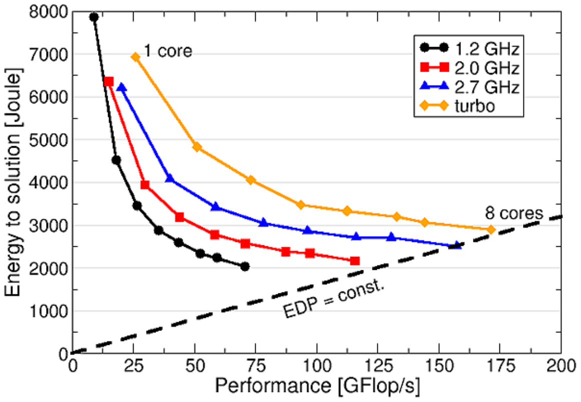


Fig. 6. Measured power and performance. The dashed line shows points of constant EDP fitted to the 2.7 GHz result with 8 cores, which represents the optimal operating point and is predicted by the model. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-140393.)

and optimization are the reuse of the instrumentation framework, and the traceability of the outcome of an optimization.

One example for such an integration is Score-P. It provides an Online Access mechanism which is enabled by an additional user instrumentation to the application that should be analyzed or optimized [30]. Periscope [17] and the Periscope Tuning Framework PTF [31] implement an online analysis for automatically tuning the energy consumption. The tool is currently developed in the AutoTune [31] project that aims for performance and energy optimizations. In their current approach Navarrete et al. [34] use an instrumented code to find an optimal processor frequency setting for the energy-to-solution of applications.

Schöne and Molka also increase the energy efficiency of applications using performance measurement tools. In [38] they use the VampirTrace framework and follow a 2-step approach. First, they instrument the application and measure hardware performance counters at a per region scale. Second, they use this information and the existing instrumentation to adjust hardware and software settings at runtime per region. With the inclusion of wattmeters in VampirTrace, Schöne and Molka validate their setup choices with another measurement run, as shown in Fig. 7. They propose multiple optimizations like frequency scaling, concurrency throttling and prefetcher settings.

8. Constant-power tuning considerations

The forgoing sections have demonstrated how algorithmic approaches to saving energy (and to a lesser extent, saving power) have developed into a mature field of study with significant impact on hardware design. However, as supercomputers continue to move to exascale, the primary constraint on performance becomes the bound on electrical power available to the system. Diverging from this bound, either above or below, will incur significant cost. The question turns from one of energy efficiency to one of optimizing performance under a power bound. Changing the focus from energy to performance has several profound effects on the design of extreme-scale systems.

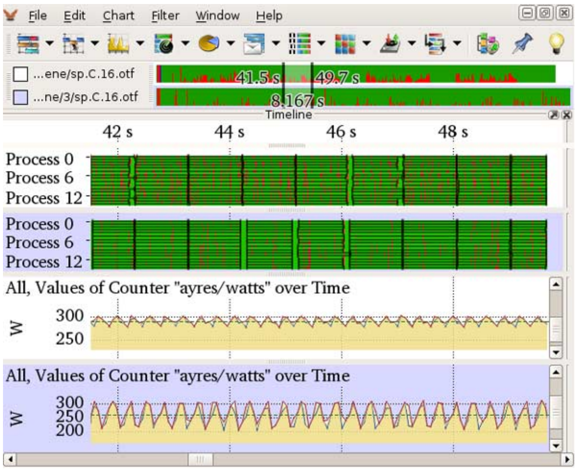


Fig. 7. Vampir Comparison View: Normal vs. energy efficient execution of the MPI parallel NPB benchmark SP on a dual-socket Sandy Bridge system. Energy efficiency is provided by the usage of DVFS via `libadapt` and VampirTrace. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-140393.)

Hardware overprovisioning. Given that utilization should be measured from the scarce resource, the utilization of existing machines is (correctly) measured in terms of node-hours. For these designs, machines were provisioned to have power sufficient to run all nodes at peak performance (*worst-case provisioning*). If this provisioning model was to carry over to extreme-scale designs, the node count would be limited by the assumption that nodes would always draw peak power. Such a system would never exceed its power limit, but as most scientific codes do not consistently require peak node power, a portion of the allocated power would be consistently unused. From a node-hours point of view the machine may be fully utilized, but measuring percent power consumption tells a different story. To guarantee full power utilization, designs should instead be *hardware-overprovisioned*. The system would have more nodes than can be run at full power until the power bound, and a combination of the job scheduler and runtime system will limit per-node power consumption so that the full allocation of power is used at all times. This may require that a subset of nodes be idled from time to time, but as power (not nodes) is the bottleneck resource, this outcome should maximize performance.

Scheduling. Job scheduling over time and node counts is already a known NP-hard problem [47]. Adding a user-requested power allocation adds another level of complexity. Users need to understand how power, node counts and performance interact with their code in order to request an appropriate amount of power for a given job. The scheduler must release the job when both a sufficient number of nodes and power is forecast to be available for the expected duration of the job. When a job completes, the scheduler will be required to decide whether the newly-freed nodes and power should be used to start another job in the queue, or whether some or all of the newly-freed nodes should be put into a low-power sleep state and the power distributed to already-running jobs. The scheduler may also be required to handle a fluctuating system-wide power bound, either due to dependence on variable sources of energy (such as wind or solar) or hour-to-hour fluctuations in the pricing of electricity.

Node configuration. With the introduction of Turbo frequencies on newer x86 processors, performance modeling became significantly more difficult. Existing models, such as those described earlier in this paper, rely on a simplification that performance (in the absence of other bottlenecks) increases linearly with core count if no bottleneck is present. Turbo mode turns this on its head: higher frequencies are available, but only if few cores are used. Memory bus bottlenecks are common for scientific codes, and adding either more cores or higher frequencies once the bus is saturated burns power without any increase in performance. Adding a power bound to this mix complicates performance modeling still further: should per-node power be reduced in order to bring up additional nodes, or should fewer, hotter nodes be scheduled? The decision whether to use multi-threading, GPU accelerators, or vector units takes this problem well beyond what existing models can handle.

Load imbalance. Despite years of research, real-world applications continue to suffer from load imbalance. Rebalancing power within a running job rather than rebalancing work distributions may be more efficient in terms of both performance and energy. Taken in combination, we expect data balancing only needs to get within 10% of the ideal for power balancing to approach optimal execution.

Getting to an exaflop within 20 MW is largely a hardware problem. Making the best use of such a system in a production environment however is a system software problem, and one that will need to be solved not only for exascale, but for all systems going forward into the future.

9. Conclusion and outlook

In this paper we have described some aspects of state-of-the-art energy efficiency tuning. We have presented common energy efficiency metrics and provided an overview of energy efficiency analysis tools. We have discussed what problems arise from including such information and how different tools make use of the metrics to provide users with hints how energy is wasted in parallel applications. The integration of energy and power metrics into performance measuring infrastructures however is only a first step to tuning. The next step will be the integration of tuning cycles and performance measurement. The reusage of sampling and instrumentation infrastructures provides a convenient front end for such an integration. We have discussed first attempts for such an integration in previous papers.

The Online Access feature of Score-P provides a promising interface for auto-tuning performance and energy. The future plans of the energy aware AutoTune plug-ins are related to the optimization of the search algorithm by reducing the search space, which should lead to a better performance. Currently, the search algorithm consists of an independent and exhaustive search which tests all available frequencies and governors. The next versions of the plugin will integrate a heuristic search to focus the search just on a minimized set of frequencies and governors. The heuristic will be based on a energy model which is ongoing development.

The measurement of power consumption however is still an open field for engineering and research. Most power measurement devices and infrastructures provide a limited temporal, spatial, or reading resolution. Also external influences like OS noise and temperature issues influence the quality and reproducibility of power measurements.

Another part of our future work is the development of performance counter based energy models. As was said before, measuring the power consumption of tasks at a fine granular level is currently hard to achieve. Thus, a model that does not rely on asynchronous data with low resolution is inevitable to understand the energy efficiency at fine grained region level. Creating such a model however provides certain pitfalls.

Acknowledgements

This work has been funded by the Bundesministerium für Bildung und Forschung via the research projects CoolSilicon (BMBF 13N10186), Score-E (BMBF 01IH13001C) and FEPA (BMBF 01IH1300 9A) and the European projects FP7-318793 "Exa2 Green" and FP7-ICT-2011-7 "AutoTune". Further funding was received from the state of North-Rhine Westphalia ("Anschubfinanzierung zum Aufbau des Exascale Innovation Center (EIC)").

References

[1] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey and N.R. Tallent, Hpctoolkit: tools for performance analysis of optimized parallel programs, *Concurrency and Computation: Practice and Experience* **22**(6) (2010), 685–701.

[2] Advanced configuration and power interface (acpi) specification, revision 5.0, December 2011.

[3] J.I. Aliaga, M. Bollhöfer, A.F. Martín and E.S. Quintana-Ortí, Exploiting thread-level parallelism in the iterative solution of sparse linear systems, *Parallel Computing* **37**(3) (2011), 183–202.

[4] P. Alonso, R.M. Badia, J. Labarta, M. Barreda, M.F. Dolz, R. Mayo, E.S. Quintana-Orti and R. Reyes, Tools for power-energy modelling and analysis of parallel scientific applications, in: *41st International Conference on Parallel Processing (ICPP)*, 2012, pp. 420–429.

[5] A. Auweter, A. Bode, M. Brehm, L. Brochard, N. Hammer, H. Huber, R. Panda, F. Thomas and T. Wilde, A case study of energy aware scheduling on supermuc, in: *International Supercomputing Conference (ISC) Proceedings 2014*, 2014, accepted for publication.

[6] S. Barrachina, M. Barreda, S. Catalán, M.F. Dolz, G. Fabregat, R. Mayo and E.S. Quintana-Ortí, An integrated framework for power-performance analysis of parallel scientific workloads, in: *3rd Int. Conf. Smart Grids, Green Communications and IT Energy-Aware Technologies*, 2013, pp. 114–119.

[7] M. Barreda, S. Catalán, M.F. Dolz, R. Mayo and E.S. Quintana-Ortí, Automatic detection of power bottlenecks in parallel scientific applications, in: *Computer Science – Research and Development*, 2013, pp. 1–9.

[8] D. Bedard, M.Y. Lim, R. Fowler and A. Porterfield, Powermon: Fine-grained and integrated power monitoring for commodity computer systems, in: *Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon)*, 2010, pp. 479–484.

[9] C. Bekas and A. Curioni, A new energy aware performance metric, *Computer Science – Research and Development* **25**(3,4) (2010), 187–195.

[10] D.M. Brooks, P. Bose, S.E. Schuster, H. Jacobson, P.N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta and P.W. Cook, Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors, *Micro, IEEE* **20**(6) (2000), 26–44.

[11] G. Contreras and M. Martonosi, Power prediction for Intel xscale reg; processors using performance monitoring unit events, in: *Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005. ISLPED'05*, August 2005, pp. 221–226.

[12] M.E.M. Diouri, M.F. Dolz, O. Glück, L. Lef'evre, P. Alonso, S. Catalán, R. Mayo and E.S. Quintana-Ortí, Solving some mysteries in power monitoring of servers: Take care of your wattmeters!, in: *Energy Efficiency in Large Scale Distributed Systems*, J.-M. Pierson, G. Da Costa and L. Dittmann, eds, Lecture Notes in Computer Science, Springer, Berlin/Heidelberg, 2013, pp. 3–18.

[13] J.J. Dongarra, Performance of various computers using standard linear equations software in a Fortran environment, *SIGARCH Comput. Archit. News* **16**(1) (1988), 47–69.

[14] W.-C. Feng and T. Scogland, The Green500 List: Year One, in: *5th IEEE Workshop on High-Performance, Power-Aware Computing (in conjunction with the 23rd International Parallel & Distributed Processing Symposium)*, Rome, Italy, May 2009.

[15] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li and K.W. Cameron, Powerpack: Energy profiling and analysis of high-performance systems and applications, *IEEE Transactions on Parallel and Distributed Systems* **21**(5) (2010), 658–671.

[16] M. Geimer, F. Wolf, B.J.N. Wylie, E. Ábrahám, D. Becker and B. Mohr, The Scalasca performance toolset architecture, *Concurrency and Computation: Practice and Experience* **22**(6) (2010), 702–719.

[17] M. Gerndt and M. Ott, Automatic performance analysis with periscope, *Concurrency and Computation: Practice and Experience* **22**(6) (2010), 736–748.

[18] B. Goel, S.A. McKee, R. Gioiosa, K. Singh, M. Bhadauria and M. Cesati, Portable, scalable, per-core power estimation for intelligent resource management, in: *IGCC*, August 2010, pp. 135–146.

[19] D. Hackenberg, T. Ilsche, R. Schöne, D. Molka, M. Schmidt and W.E. Nagel, Power measurement techniques on standard compute nodes: A quantitative comparison, in: *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 194–204.

[20] G. Hager, J. Treibig, J. Habich and G. Wellein, Exploring performance and power properties of modern multi-core chips via simple machine models, in: *Concurrency and Computation: Practice and Experience*, 2014.

[21] M. Horowitz, T. Indermaur and R. Gonzalez, Low-power digital design, in: *Low Power Electronics, 1994. Digest of Technical Papers, IEEE Symposium*, 1994, pp. 8–11.

[22] IBM systems director active energy manager. Installation and user's guide, Technical report, IBM Corporation, 2012.

[23] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual Volumes 3A, 3B, and 3C: System Programming Guide, Parts 1 and 2*, September 2013.

[24] V. Jimenez, R. Gioiosa, F.J. Cazorla, M. Valero, E. Kursun, C. Isci, A. Buyuktosunoglu and P. Bose, Energy-aware accounting and billing in large-scale computing facilities, *IEEE Micro* **31**(3) (2011), 60–71.

[25] R. Joseph and M. Martonosi, Run-time power estimation in high performance microprocessors, in: *Proceedings of the 2001 International Symposium on Low Power Electronics and Design, ISLPED'01*, ACM, New York, NY, USA, 2001, pp. 135–140.

[26] R. Jotwani, S. Sundaram, S. Kosonocky, A. Schaefer, V. Andrade, G. Constant, A. Novak and S. Naffziger, An x86-64 core implemented in 32 nm soi cmos, in: *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, 2010, pp. 106–107.

[27] M. Knobloch, M. Foszczynski, W. Homberg, D. Pleiter and H. Böttiger, Mapping fine-grained power measurements to HPC application runtime characteristics on IBM POWER7, in: *Computer Science – Research and Development*, 2013, pp. 1–9.

[28] M. Knobloch, B. Mohr and T. Minartz, Determine energy-saving potential in wait-states of large-scale parallel programs, *Computer Science – Research and Development* **27** (2012), 255–263, Record converted from VDB: 12.11.2012.

[29] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M.S. Müller and W.E. Nagel, The vampir performance analysis tool-set, in: *Tools for High Performance Computing*, M. Resch, R. Keller, V. Himmler, B. Krammer and A. Schulz, eds, Springer, Berlin/Heidelberg, 2008, pp. 139–155.

[30] A. Knüpfer, C. Rössel, D. Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W.E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. Shende, R. Tschüter, M. Wagner, B. Wesarg and F. Wolf, Score-p: A joint performance measurement runtime infrastructure for periscope, scalasca, tau, and vampir, in: *Tools for High Performance Computing 2011*, H. Brunst, M.S. Müller, W.E. Nagel and M.M. Resch, eds, Springer, Berlin/Heidelberg, 2012, pp. 79–91.

[31] R. Miceli, G. Civario, A. Sikora, E. César, M. Gerndt, H. Haitof, C. Navarrete, S. Benkner, M. Sandrieser, L. Morin and F. Bodin, Autotune: A plugin-driven approach to the automatic tuning of parallel applications, in: *Applied Parallel and Scientific Computing*, P. Manninen and P. Öster, eds, Lecture Notes in Computer Science, Vol. 7782, Springer, Berlin/Heidelberg, 2013, pp. 328–342.

[32] D. Molka, D. Hackenberg, R. Schöne and M.S. Müller, Characterizing the energy consumption of data transfers and arithmetic operations on x8664 processors, in: *Proceedings of the 1st International Green Computing Conference*, IEEE, 2010, pp. 123–133.

[33] M.S. Müller, J. Baron, W.C. Brantley, H. Feng, D. Hackenberg, R. Henschel, G. Jost, D. Molka, C. Parrott, J. Robichaux, P. Shelepugin, M. Waveren, B. Whitney and K. Kumaran, Spec omp2012 – an application benchmark suite for parallel systems using openmp, in: *OpenMP in a Heterogeneous World*, B. Chapman, F. Massaioli, M.S. Müller and M. Rorro, eds, Lecture Notes in Computer Science, Vol. 7312, Springer, Berlin/Heidelberg, 2012, pp. 223–236.

[34] C.B. Navarrete, C. Guillen, W. Hesse and M. Brehm, Autotuning the energy consumption, in: *PARCO*, Advances in Parallel Computing, IOS Press, 2014, pp. 668–677.

[35] New IBM switched and monitored family of power distribution units makes it easy to protect and manage high-availability rack-based systems, Technical report, IBM Corporation, 2010.

[36] V. Pillet, J. Labarta, T. Cortes and S. Girona, Paraver: A tool to visualize and analyze parallel code, in: *WoTUG-18*, 1995, pp. 17–31.

[37] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan and E. Weissmann, Power-management architecture of the Intel microarchitecture code-named sandy bridge, *Micro, IEEE* **32**(2) (2012), 20–27.

[38] R. Schöne and D. Molka, Integrating performance analysis and energy efficiency optimizations in a unified environment, *Computer Science – Research and Development*, 2013, pp. 1–9.

[39] R. Schöne, R. Tschüter, T. Ilsche and D. Hackenberg, The vampirtrace plugin counter interface: introduction and examples, in: *Proceedings of the 2010 Conference on Parallel Processing, Euro-Par 2010*, Springer-Verlag, Berlin/Heidelberg, 2011, pp. 501–511.

[40] M. Schulz, J. Galarowicz, D. Maghrak, W. Hachfeld, D. Montoya and S. Cranford, Open–speedshop: An open source infrastructure for parallel performance analysis, *Sci. Program.* **16**(2,3) (2008), 105–121.

[41] H. Servat, G. Llort, J. Giménez and J. Labarta, Detailed and simultaneous power and performance analysis, in: *Concurrency and Computation: Practice and Experience*, 2013.

[42] S.S. Shende and A.D. Malony, The tau parallel performance system, *Int. J. High Perform. Comput. Appl.* **20**(2) (2006), 287–311.

[43] K. Singh, M. Bhadauria and S.A. McKee, Real time power estimation and thread scheduling via performance counters, *SIGARCH Comput. Archit. News* **37**(2) (2009), 46–55.

[44] D. Terpstra, H. Jagode, H. You and J. Dongarra, Collecting performance data with papi-c, in: *Tools for High Performance Computing 2009*, Springer, 2010, pp. 157–173.

[45] J. Treibig and G. Hager, Introducing a performance model for bandwidth-limited loop kernels, in: *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, J. Dongarra, K. Karczewski and J. Wasniewski, eds, Lecture Notes in Computer Science, Vol. 6067, Springer, Berlin/Heidelberg, 2010, pp. 615–624.

[46] J. Treibig, G. Hager and G. Wellein, LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments, in: *PSTI2010, The First International Workshop on Parallel Software Tools and Tool Infrastructures*, Los Alamitos, CA, USA, IEEE Computer Society, 2010, pp. 207–216.

[47] J.D. Ullman, Np-complete scheduling problems, *Journal of Computer and System Sciences* **10**(3) (1975), 384–393.

[48] M. Wittmann, G. Hager, T. Zeiser and G. Wellein, An analysis of energy-optimized lattice-Boltzmann cfd simulations from the chip to the highly parallel level, in: *CoRR*, 2013, abs/1304.7664.

[49] X. Yang, Z. Zhou, S. Wallace, Z. Lan, W. Tang, S. Coghlan and M.E. Papka, Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems, in: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC'13*, ACM, New York, NY, USA, 2013, pp. 60:1–60:11.

[50] Z. Zhou, Z. Lan, W. Tang and N. Desai, Reducing energy costs for IBM blue gene/p via power-aware job scheduling, in: *Workshops on Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2013.